

# Teoría de Lenguajes - Segundo Parcial

Primer cuatrimestre de 2019

Apagar los celulares.

Hacer cada ejercicio en hojas separadas.

Poner nombre, número de orden y número de página en cada ejercicio.

Justificar todas las respuestas.

El examen es a libro abierto.

Se aprueba con 65 puntos.

La siguiente gramática  $G$  representa un fragmento de las expresiones de un lenguaje funcional similar a Haskell:

$G = \langle \{id, :, (, ), [, ], \{E\}, P, E \rangle$ , con  $P$ :

$E$	$\rightarrow$	$id$	
		$E E$	Aplicar función
		$[ ]$	Lista vacía
		$E : E$	Agregar elemento al frente de una lista
		$( E )$	

La operación de construcción de listas  $:$  es asociativa a derecha y tiene menor precedencia que la aplicación de funciones, que es asociativa a izquierda. Es decir que la cadena:

$a : b c [] : []$

Debe interpretarse como:

$a : ((b c []) : [])$

1. (34 pts)

a) Dar la tabla SLR para  $G$  señalando todos los conflictos que tenga.

b) Resolver los conflictos eligiendo en cada caso una de las entradas de la tabla de manera que el lenguaje aceptado sea  $L(G)$  y el árbol de derivación resultante respete las precedencias y asociatividades descriptas.

2. (33 pts) Dar una gramática extendida que sea ELL(1) y que genere  $L(G)$ .

3. (33 pts) Convertir a  $G$  en una *gramática de atributos* que sintetice en un atributo de tipo *string* una expresión equivalente a la original pero sin paréntesis innecesarios. El no terminal  $id$  tiene un atributo **nombre** de tipo *string*. Asumir que el árbol de derivación se armará según las precedencias descriptas. Por ejemplo, para la cadena de entrada:

$a (b : []) : (((c d) : []) : [])$

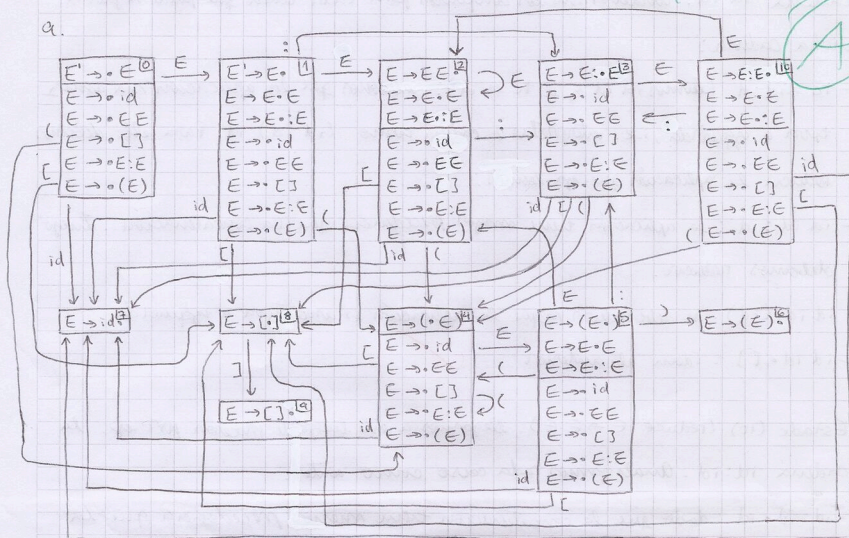
se debe sintetizar:

$a (b : []) : (c d : []) : []$

FECHA	1	2	3	T
	39	33	33	100

1)

a.



	id	:	(	)	[	]	$\Phi$	E	Producciones
0	s7		s4		s8			1	
1	s7	s3	s4		s8		accept	2	$E \rightarrow id$ (1)
2	r(2)/s7	r(2)/s3	r(2)/s4	r(2)	r(2)/s8		r(2)	2	$E \rightarrow EE$ (2)
3	s7		s4		s8			10	$E \rightarrow []$ (3)
4	s7		s4		s8			5	$E \rightarrow E:E$ (4)
5	s7	s3	s4	s6	s8			2	$E \rightarrow (E)$ (5)
6	r(5)	r(5)	r(5)	r(5)	r(5)		r(5)		
7	r(1)	r(1)	r(1)	r(1)	r(1)		r(1)		
8									
9	r(3)	r(3)	r(3)	r(3)	r(3)		r(3)		
10	r(4)/s7	r(4)/s3	r(4)/s4	r(4)	r(4)/s8		r(4)	2	

Primeros(E) = {id, [, {

Siguietes(E) = { $\Phi$ , id, [, (, :, )}

Los conflictos son los sub-rayados.



b.

Estado (2) (reduce  $E \rightarrow EE$ ): llegamos a este luego de procesar, por ej., la cadena  $id\ id$ . Analizamos los conflictos para cada token que puede seguir a dicha cadena.

- $id\ id \cdot id$  (leemos  $id\ id$  y el  $te$  es  $id$ ): queremos que la aplicación sea asociativa a izquierda, i.e. interpretar la cadena como  $(id\ id)\ id$ . Para ello debemos reducir la aplicación que ya leemos.
- $id\ id \cdot :$  la aplicación tiene mayor precedencia que la concatenación. Luego debemos reducir.
- $id\ id \cdot (id)$ : se debe reducir porque la aplicación es asociativa a izquierda.
- $id\ id \cdot [ ]$ : ídem al anterior

Estado (10) (reduce  $E \rightarrow E:E$ ): llegamos a este luego de procesar por ej. la cadena  $id:id$ . Analizamos cada caso como antes.

- $id:id \cdot id$ : dado que la concatenación tiene menos precedencia que la aplicación, debemos hacer shift para que se reduzca primero la aplicación.
- $id:id \cdot ::id$ : como la concatenación es asociativa a derecha hacemos shift para que reduzcan primero las concatenaciones de la derecha.
- $id:id \cdot (id)$ : la aplicación tiene mayor precedencia. Luego debemos hacer shift para reducir a esta primero.
- $id:id [ ]$ : ídem al anterior

Luego la tabla para estos dos estados queda:

	id	:	(	)	:	[	]	\$
2	r(2)	r(2)	r(2)	r(2)	r(2)			r(2)
10	s7	s3	s4	r(4)	s8			r(4)

2)

Doy una GLC que resuelve los conflictos de asociatividad de  $G$ , pero que enfuerce siempre la asociatividad a derecha para no tener recursión a izquierda.

$G' = \langle \Sigma, \{id, \cdot, (, ), \cdot, \cdot\}, \{E, F, T\}, P, E \rangle$  con  $P$ :

$$E \rightarrow FE \mid F$$

$$F \rightarrow T \mid F \mid T$$

$$T \rightarrow id \mid ( \mid (E)$$

El problema de esta gramática es que tiene producciones de un mismo no terminal que tienen un mismo prefijo (ej:  $E \rightarrow FE \mid F$ ). Para resolver esto factorizo a izquierda:

$G' = \langle \Sigma, \{E, X, F, Y, T\}, P, E \rangle$  con  $P$ :

$$E \rightarrow FX$$

$$X \rightarrow E \mid \lambda$$

$$F \rightarrow TY$$

$$Y \rightarrow :F \mid \lambda$$

$$T \rightarrow id \mid ( \mid (E)$$

Veamos que  $G'$  es  $LL(1)$ :

	Primeros	Siguientes		SD	
E	id, (, (	$\emptyset, )$	$\checkmark$	$X \rightarrow E$	Primeros( $E$ ) = {id, (, (}
X	id, (, (	$\emptyset, )$	$\checkmark$	$X \rightarrow \lambda$	Siguientes( $x$ ) = { $\emptyset, )$ }
F	id, (, (	id, (, (, $\emptyset$ , )	$\checkmark$	$Y \rightarrow :F$	Primeros( $:F$ ) = { $\emptyset$ }
Y	:	id, (, (, $\emptyset$ , )	$\checkmark$	$Y \rightarrow \lambda$	Siguientes( $Y$ ) = {id, (, (, $\emptyset$ , )}
T	id, (, (	id, (, (, $\emptyset$ , )	$\checkmark$	$T \rightarrow id$	id
			$\checkmark$	$T \rightarrow ($	(
			$\checkmark$	$T \rightarrow (E)$	(

No hay conflictos para ninguna producción. Luego  $G'$  es  $LL(1)$ .

Como, haciendo un abuso de notación, podemos interpretar a las producciones

$X \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$  como una producción de una gramática extendida

en la que  $\alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$  es una expresión regular, podemos decir que  $G'$  es una gramática extendida. Luego, como es  $LL(1)$ , es  $ELL(1)$ .



3)

Las expresiones  $id$  y  $[]$  nunca necesitan paréntesis, mientras que la aplicación y la concatenación pueden necesitarlos en algunos casos. Esto casos son:

- Para la aplicación: cuando se hace a la derecha de otra aplicación (ej:  $id(id\ id)$ ). ✓

- Para la concatenación: cuando se hace a la izquierda de otra concatenación (ej:  $(id:id):id$ ) o cuando está en una aplicación (ej:  $id(id:id)$ ). ✓

La idea para generar la gramática de atributos es entonces identificar estos casos para poner los paréntesis cuando son necesarios. Para ello usamos dos atributos sintetizados  $E.app$  y  $E.concat$  de tipo boolean que indican si la expresión es una aplicación o concatenación respectivamente. ✓

$E \rightarrow id \quad \{ E.app = false; E.concat = false; E.exp = id.nombre \}$

$E \rightarrow [] \quad \{ E.app = false; E.concat = false; E.exp = "[]" \}$

$E \rightarrow (E_1) \quad \{ E.app = E_1.app; E.concat = E_1.concat; E.exp = E_1.exp \}$

$E \rightarrow E_1 E_2 \quad \left\{ \begin{array}{l} E.app = true; E.concat = false; \\ E.exp = IF(E_1.concat, "(" + E_1.exp + ")", E_1.exp) + \\ IF(E_2.concat \vee E_2.app, "(" + E_2.exp + ")", E_2.exp) \end{array} \right\}$  ✓

$E \rightarrow E_1 : E_2 \quad \left\{ \begin{array}{l} E.app = false; E.concat = true; \\ E.exp = IF(E_1.concat, "(" + E_1.exp + ")", E_1.exp) + ":" + E_2.exp \end{array} \right\}$  ✓

$E.exp$  es sintetizado de tipo string y cuando la expresión sin paréntesis innecesarios ✓